

Bachelor's thesis

Degree programme in Information Technology

NINFOS 12

2016

Minh Nguyen

MACHINE LEARNING: DEVELOPING AN IMAGE RECOGNITION PROGRAM

– with Python, Scikit Learn and OpenCV



BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme in Information Technology | Internet Technology

2016 | 50

Instructor: Patric Granholm

Minh Nguyen

MACHINE LEARNING: DEVELOPING AN IMAGE RECOGNITION PROGRAM

- with Python, Scikit Learn and OpenCV

Machine Learning is one of the most debated topic in computer world these days, especially after the first Computer Go program has beaten human Go world champion. Among endless application of Machine Learning, image recognition, which problem is processing enormous amount of data from dynamic input.

This thesis will present the basic concept of Machine Learning, Machine Learning algorithms, Python programming language and Scikit Learn – a simple and efficient tool for data analysis in Python. To demonstrate all of these concepts, a small program has been created whose function was to first process a dataset which contain a number of face images, then feed it through a machine learning algorithm and finally using live detection from a webcam to give the live input for the program to predict weather or not the person is smiling.

KEYWORDS:

Machine Learning, Python, Scikit Learn, OpenCV

CONTENT

1 INTRODUCTION	6
2 MACHINE LEARNING	8
3 SUPPORT VECTOR MACHINE (SVM)	15
3.1 SVM definition	15
3.2 Separating Hyperplane and Margin	17
3.3 Finding the Optimal Hyperplane	20
4 OPENCV AND SCIKIT LEARN	29
4.1 OpenCV library	29
OpenCV introduction	29
OpenCV Face Detection using Haar Cascades	30
4.2 Sci-kit Learn	32
5 THE SMILE RECOGNITION PROGRAM	34
5.1 The preparation of the dataset	34
5.2 Training the classifier	37
5.3 Smile Detection using live webcam	40
6 CONCLUSION	43
REFERENCES	44

APPENDICES

Appendix 1. Simple program to take user respond for classifying data

Appendix 2. Display the statistic of the dataset

Appendix 3. Classifier trainer

Appendix 4. Testing the classifier

Appendix 5. Running classifier through video loop

FIGURES

Figure 1: Visualized SVM.	15
Figure 2: Example dataset.	16
Figure 3: Example dataset with hyperplane.	16
Figure 4: Example dataset with Hyperplane and Margin.	17
Figure 5: Example dataset 2.	20
Figure 6: Example dataset 2 with Optimal Hyperplane.	21
Figure 7: Two Hyperplanes satisfying the constraints.	24
Figure 8: Two Hyperplanes also satisfying the constraints.	25
Figure 9: H1 does not satisfy the first constraint.	25
Figure 10: Finding margin m	27
Figure 11: Simple interface classification program	35
Figure 12: Smile vs. no smile statistic	36
Figure 13: Smiling faces	37
Figure 14: Not smiling faces	37
Figure 15: Highlighted face region	39
Figure 16: Extracted face	40
Figure 17: Live detection - smile	41
Figure 18: Live detection - no smile	42

EQUATIONS

Equation 1: Transformation from general hyperplane equation to a line equation	18
Equation 2: Calculation of the margin.....	19
Equation 3: Transformation of hyperplane equation in two dimension.....	22
Equation 4: Combination of the constraints.....	26
Equation 5: Calculation of the margin.....	28

PICTURES

Picture 1: Stanford Cart. Source: www.computerhistory.org .	9
Picture 2: Kasparov Vs. Deep Blue. Source: www.forbes.com .	10
Picture 3: Google's AlphaGO vs. Lee Sedol. Source: www.flashofgold.com .	11
Picture 4: Haar features. Source: http://opencv-python-tutroals.readthedocs.io/	30
Picture 5: Haar features in face recognition. Source: http://opencv-python-tutroals.readthedocs.io/	32

LIST OF ABBREVIATIONS (OR) SYMBOLS

SVM	Support Vector Machine
OpenCV	Open Source Computer Vision
Scikit Learn	Machine Learning Library
MMX	Meaningless initialism trademarked by Intel
SSE	Streaming SIMD Extensions
Adaboos	Adaptive Boosting

1 INTRODUCTION

Machine Learning is not a brand new topic among data scientists but it has been increasingly gaining popularity these days. One of the reason for Machine Learning to shine is that, the old tradition programming methods are no longer suitable for the ever-changing world of information nowadays. Dealing with gigantic amount of information which is always shifting, the old tradition programming methods simply do not have the ability to adapt because they were designed to work with only particular set of problems. Machine Learning, on the other hand, can learn from previous computations to reproduce reliable, repeatable decisions and result without changing the code.

So, what is Machine Learning? There are many debates revolving around the definition of Machine Learning. From the very first definition of Machine Learning, "Machine Learning: Field of study that gives computers the ability to learn without being explicit programmed." [1], till the recent definition, "Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measure by P , improves with experience E ." [2], we can conclude that Machine Learning is a type of Artificial Intelligent whose ability is teaching themselves to grow and change when exposed to new data, similar to the way human learn from experiences.

There are several ways or algorithms to program computer to learn by itself but typically, they can be categorized into three broad categories: Supervised Learning, Unsupervised Learning and Reinforcement. We will take a brief look at all the categories in the later part in this thesis.

For the purpose of demonstration, the program that have been created used the Supervised Learning method. A set of data (collection of face images) first will be processed and divided into two main groups, then it will be passed on a classifier (algorithm to help machine learn from data) to train the Machine Learning program, finally, using webcam and a small face recognition to extract the face image and put through the Machine Learning program to identify if the person is smiling or not.

This thesis will cover these topics: Chapter 2 will introduce the overview of learning machine, Chapter 3 will talk about Support Vector Machine (SVM) which will be used later on the program, Chapter 4 is spent to mention about OpenCV (Open Source Computer Vision) which is the library of programming functions aimed at real-time computer vision and is used to provide input for the program, finally, Chapter 5 will present the structure of the Smile Recognition Program and explain the code behind it.

2 MACHINE LEARNING

Machine learning is a branch of AI (Artificial Intelligent) focusing on systems that can improve themselves from the environment. The goal is to generalize this training and give the machine the ability to adapt to new environment. In order to understand the evolution of Machine Learning, we will take a look at brief history of Machine Learning through these timelines:

1950 — Alan Turing creates the “Turing Test” to determine if a computer has real intelligence. To pass the test, a computer must be able to fool a human into believing it is also human. [3]

1957 — Perceptron Algorithm was invented at the Cornell Aeronautical Laboratory by Frank Rosenblatt, was the first Artificial Network designed for image recognition. [4]

1959 — Arthur Samuel wrote the first Machine Learning program to play (and win) checkers. The program chose its move based on a minimax strategy, meaning it made the move that optimized the value of this function, assuming that the opponent was trying to optimize the value of the same function from its point of view. By the mid 1970's, his program was beating capable human players. [5]

1967 — The “nearest neighbor” algorithm was written, used to determine a solution to the travelling salesman problem. In it, the salesman starts at a random city and repeatedly visits the nearest city until all have been visited. It quickly yields a short tour, but usually not the optimal one. [6]

1979 — Students at Stanford University invent the “Stanford Cart” which can navigate obstacles in a room on its own. [7]



Picture 1: Stanford Cart. Source: www.computerhistory.org.

1985 — Terrence Sejnowski invented NETtalk which is an artificial neural network that can be trained to pronounce English words, consists of about 300 neurons arranged in three layers - an input layer, which reads the words, an output layer, which generates speech sounds, or phonemes, and a middle, "hidden layer," which mediates between the other two. [8]

1997 — IBM's Deep Blue beats Chess Grandmaster Garry Kasparov in a six-game match. [9]



Picture 2: Kasparov Vs. Deep Blue. Source: www.forbes.com.

2006 — “Deep Learning” was mentioned as new algorithms that let computers see and distinguish objects and text in images and videos. [10]

2016 — In October 2015, Google’s AlphaGO became the first computer program to beat a professional human Go player.

So in just around 70 years, Machine Learning algorithms has evolved, especially in the beginning of 21st century, in a faster pace than ever. Along with this evolution, more and more problems are being solved, from the spam filter in email box to a predictive modelling high-level synthesis design in space exploration, it even saves lives in the application of protein analysis. So how does Machine



Picture 3: Google's AlphaGO vs. Lee Sedol. Source: www.flashofgold.com.

Learning do all the tasks which were impossible to be done by the old hard-code algorithms?

Take Supervised Learning for example, which will be used later for the program in this thesis. Supervised Learning is the machine learning task of inferring a function from supervised training data. The training data consist of a set of training examples. In Supervised Learning, each example is a pair consisting of an input object (vector) and a desired output value. A supervised learning algorithm analyzes the training data and produces an inferred function, which is called a classifier (if the output is discrete) or a regression function (if the output is continuous). The inferred function should predict the correct output value for any valid input object. This requires the learning algorithm to generalize from the training data to unseen situations in a reasonable way. In human or animal, these tasks are often referred to as concept learning.

The Supervised Learning method consists of these steps: [11]

1. Determine the type of training examples. For instance, the example might be a single handwritten character, an entire handwritten word, or an entire line of handwriting.
2. Collect a training set. The training set needs to be representative of the real-world use of the function. Thus, a set of input objects is gathered and

corresponding outputs are also gathered, either from human experts or from measurements.

3. Determine the input feature representation of the learned function. The accuracy of the learned function depends strongly on how the input object is represented. Generally, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, but also should contain enough information to accurately predict the output.
4. Determine the structure of the learned function and corresponding learning algorithm. For example, support vector machines or decision trees could be used.
5. Run the learning algorithm on the gathered training set. Some supervised learning algorithms require the user to determine certain control parameters. These parameters may be adjusted by optimizing performance on a subset (called a validation set) of the training set, or via cross-validation.
6. Evaluate the accuracy of the learned function. After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set.

Going through these steps, Supervised Machine Learning will face some major issues:

- Bias-variance tradeoff:

Learning algorithm use mathematical or statistical models whose “error” can be split into two main components: reducible and irreducible error. Irreducible error is associated with a natural variability in a system. Meanwhile, reducible error, can be and should be minimized further to maximize accuracy. Reducible error can be further decomposed into “error due to squared bias” and “error due to variance”. Error due to Bias can be defined as the difference between the expected (or average) prediction of the model and the correct value prediction. The error due to variance can be taken as the variability of a model prediction for a given data point. The variance is how much the predictions for a given point vary between different realizations of the model. Models that exhibit small

variance and high bias underfit the truth target. Models that exhibit high variance and low bias overfit the truth target. The “tradeoff” between bias and variance can be viewed in this manner – a learning algorithm with low bias must be “flexible” so that it can fit the data well. But if the learning algorithm is too flexible (for instance, too linear), it will fit each training data set differently, and hence have high variance. A key characteristic of many supervised learning methods is a built-in way to control the bias-variance tradeoff (either automatically or by providing a bias/variance parameter that the user can adjust). [12]

- Training data size and function complexity

The amount of training data and the complexity of the function are relative. If the true function is simple, then an “inflexible” learning algorithm with high bias and low variance will be able to learn it from a small amount of data. If the true function is highly complex, then the function will only be learnable from a very large amount of training data and using a “flexible” learning algorithm with low bias and high variance. Good learning algorithms therefore automatically adjust the bias/variance tradeoff based on the amount of data available and the apparent complexity of the function to be learned.

- Dimensionality of the input space

If the input feature vectors have very high dimension, the learning problem can be difficult even if the true function only depends on a small number of those features. This is because the many “extra” dimensions can confuse the learning algorithm and cause it to have high variance. Hence, high input dimensionality typically requires tuning the classifier to have low variance and high bias. To solve this issue, irrelevant features could be manually removed from the input data, this is likely to improve the accuracy of the learned function. In addition, there are many algorithms for feature selection that seek to identify the relevant features and discard the irrelevant ones. This is an instance of the more general strategy of dimensionality reduction, which seeks to map the input data into a lower-dimensional space prior to running the supervised learning algorithm. [13]

- Noise in the output values

Another issue is the degree of noise in the desired output values. If the desired output values are often incorrect (because of human error or sensor errors), then the learning algorithm should not attempt to find a function that exactly matches the training examples. Attempting to fit the data too carefully leads to overfitting. Overfitting can happen even when there are no measurement errors (stochastic noise) if the function is too complex for the learning model. In such a situation that part of the target function that cannot be modeled "corrupts" the training data - this phenomenon has been called deterministic noise. When either type of noise is present, it is better to go with a higher bias, lower variance estimator.

The next chapter will introduce Support Vector Machine, one of the most popular learning algorithm in supervised learning models and which will be used for the program in this thesis. Understanding the mechanism of the algorithm is crucial to build and implement to a machine learning program. [14]

3 SUPPORT VECTOR MACHINE (SVM)

In data analytics or decision sciences most of the time we come across the situations where we need to classify our data based on a certain dependent variable. To support the solution for this need there are multiple techniques which can be applied; Logistic Regression, Random Forest Algorithm, Bayesian Algorithm are a few to name. SVM is a machine learning technique to separate data which tries to maximize the gap between the categories (aka Margin).

3.1 SVM definition

Support Vector Machine (SVM) is a classifier method that performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different class labels.

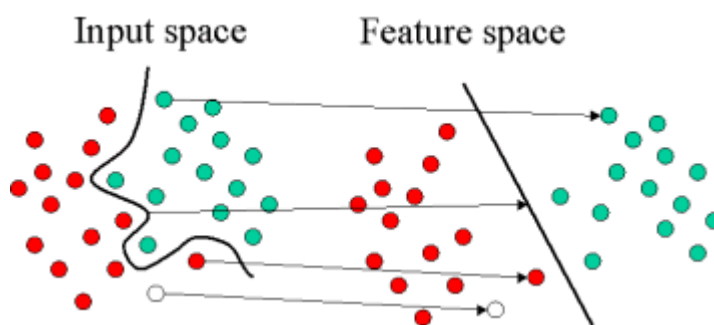


Figure 1: Visualized SVM.

The goal of a support vector machine is to find the optimal separating hyperplane which maximizes the margin of the training data. SVM can be classified as a supervised learning algorithm since it needs training data to train the classifier. For example, we have the training data shown in Figure 2:

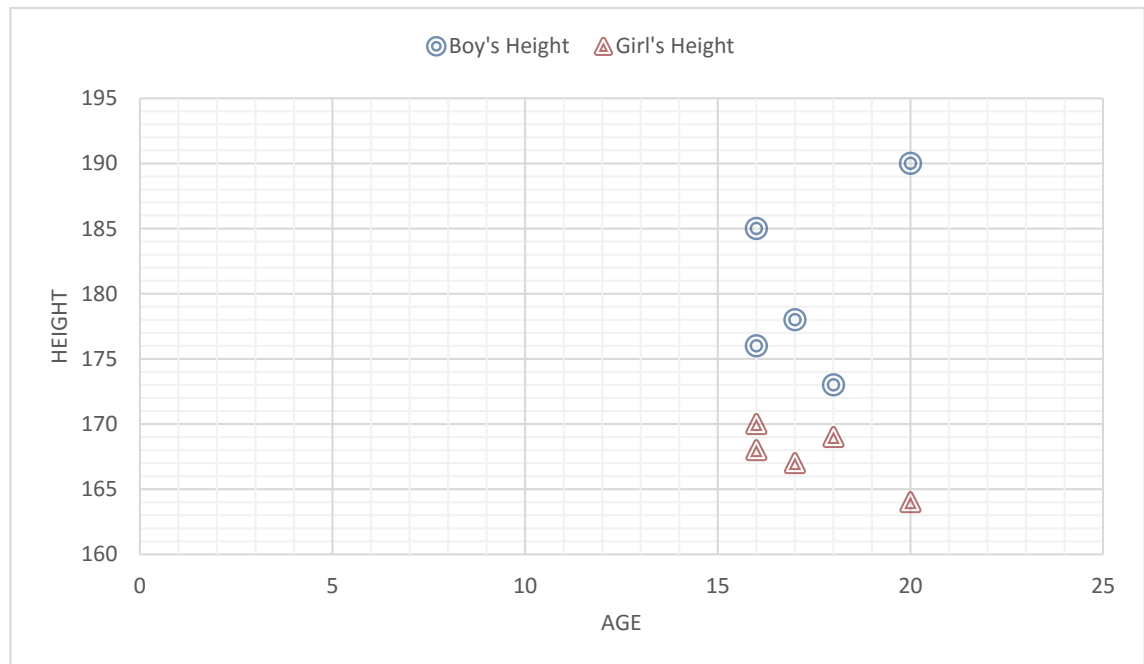


Figure 2: Example dataset.

Boy's height and Girl's height was plotted, and there is a way to distinguish between boy and girl based on their age and height. With this data, SVM will allow us to answer the question: Given a particular data point (age and height), what is the gender of the person? For instance, if the person is 180cm tall and at the age of 18, is that a boy or a girl?

Just by looking at the plot, it is possible to separate the data by tracing a line which put all the data points representing male gender above the line and female gender below the line. Such a line is called a "Separating Hyperplane".

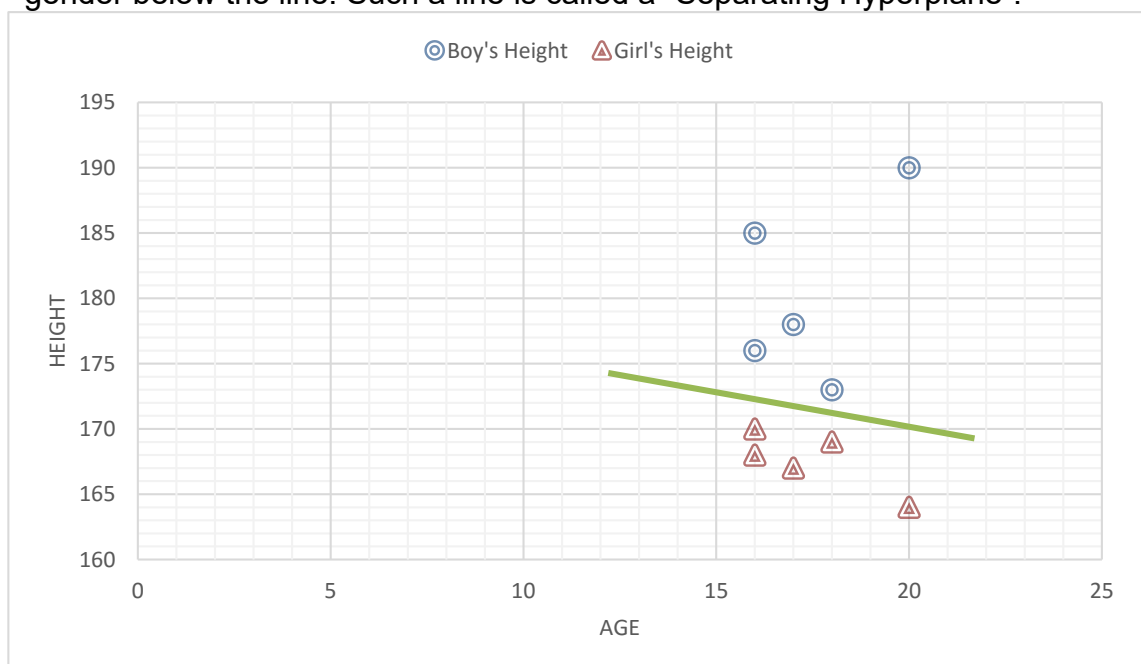


Figure 3: Example dataset with hyperplane.

Separating Hyperplane does not necessary be just a line. In one dimension, a Hyperplane is a dot. In two dimensions, it is a line. In three dimensions, it becomes a plane. Finally, in multiple dimensions, it is Hyperplane.

3.2 Separating Hyperplane and Margin

In order to correctly classify training data and make better decision for unseen data, the objective of SVM is to find the optimal Separating Hyperplane. To achieve this goal, a helper called the margin is a useful tool to assist finding Separating Hyperplane. Given a particular hyperplane, the distance between the hyperplane and the closest data point can be calculated. Margin is the double of this value. Basically, margin is the empty space, where there is no data point inside the margin.

As a rule of thumb, the optimal hyperplane will be the one whose margin will be the biggest. So the objective of the SVM is to find the optimal separating hyperplane which maximizes the margin of the training data.

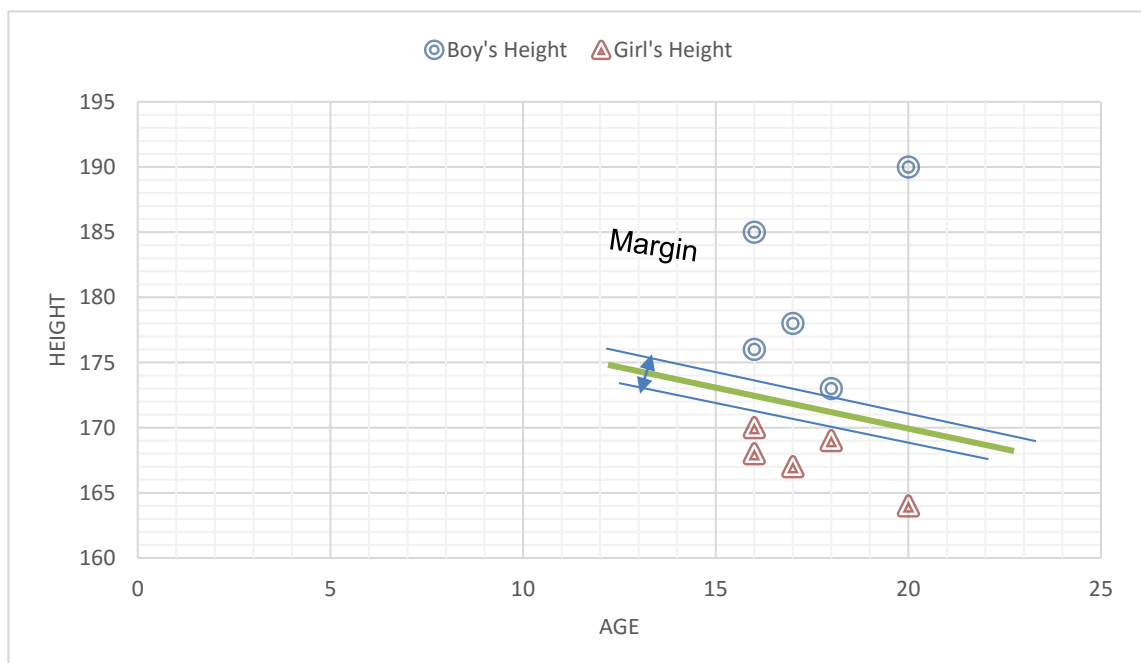


Figure 4: Example dataset with Hyperplane and Margin.

For in depth understanding about SVM, the mathematical explanation will be present in this section.

SVM – Support **Vector** Machine, mathematically, a **vector** is an object that has both a magnitude and a direction. As mentioned, the goal of SVM is to find the optimal separating hyperplane which maximized the margin of the training data.

In two dimensions, the equation of a 2 dimensional hyperplane (which is a line) is: $y = ax + b$. However in three or more dimensions, the equation of a hyperplane can be generalized as: $\vec{w}^T \cdot \vec{x} = 0$ which is the scalar product between 2 vectors, where \vec{w} is the normal vector to the hyperplane. In fact, we can denote $y = ax + b$ or $y - ax - b = 0$ as the scalar product of two vectors:

$$\vec{w}^T = (-b, -a, 1) \text{ and } \vec{x} = (1, x, y)$$

$$\vec{w}^T \cdot \vec{x} = -b \times 1 + (-a) \times x + 1 \times y$$

$$\vec{w}^T \cdot \vec{x} = y - ax - b$$

Equation 1: Transformation from general hyperplane equation to a line equation

The example presented in Figure 4 will unveil the way to compute the distance from a point to the hyperplane so that we can calculate the margin.

The goal here is to find the distance between the nearest point ($A = (3, 3)$) and the hyperplane $y = -2x$ or in another word, it is the magnitude of vector \vec{b} ($|\vec{b}|$) which is the projection of vector \vec{a} (the vector between the origin and A) onto \vec{w} (\vec{w} is the normal vector to the hyperplane)

Starting with two vectors: $\vec{w}(2, 1)$ and $\vec{a}(3, 3)$

$$|\vec{w}| = \sqrt{2^2 + 1^2} = \sqrt{5}$$

Let the vector \vec{u} be the unit vector in the direction of \vec{w}

$$\vec{u} = \left(\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}} \right)$$

\vec{b} is the orthogonal projection of \vec{a} onto \vec{w} so:

$$\vec{b} = (\vec{u} \cdot \vec{a})\vec{u}$$

$$\vec{b} = \left(3 \times \frac{2}{\sqrt{5}} + 3 \times \frac{1}{\sqrt{5}} \right) \vec{u}$$

$$\vec{b} = \left(\frac{6}{\sqrt{5}} + \frac{3}{\sqrt{5}} \right) \vec{u}$$

$$\vec{b} = \frac{9}{\sqrt{5}} \vec{u}$$

$$\vec{b} = \left(\frac{9}{\sqrt{5}} \times \frac{2}{\sqrt{5}}, \quad \frac{9}{\sqrt{5}} \times \frac{1}{\sqrt{5}} \right)$$

$$\vec{b} = \left(3\frac{3}{5}, 3\frac{3}{5} \right)$$

$$|\vec{b}| = \sqrt{\left(3\frac{3}{5} \right)^2 + \left(3\frac{3}{5} \right)^2} = 9\sqrt{\frac{1}{5}}$$

Finally, the margin can be calculated by:

$$margin = 2|\vec{b}| = 18\sqrt{\frac{1}{5}}$$

Equation 2: Calculation of the margin

However, this is not the optimal hyperplane. According to the definition, the optimal hyperplane must be the one which maximizes the margin of the training data. Finding the biggest margin is the same as finding optimal hyperplane.

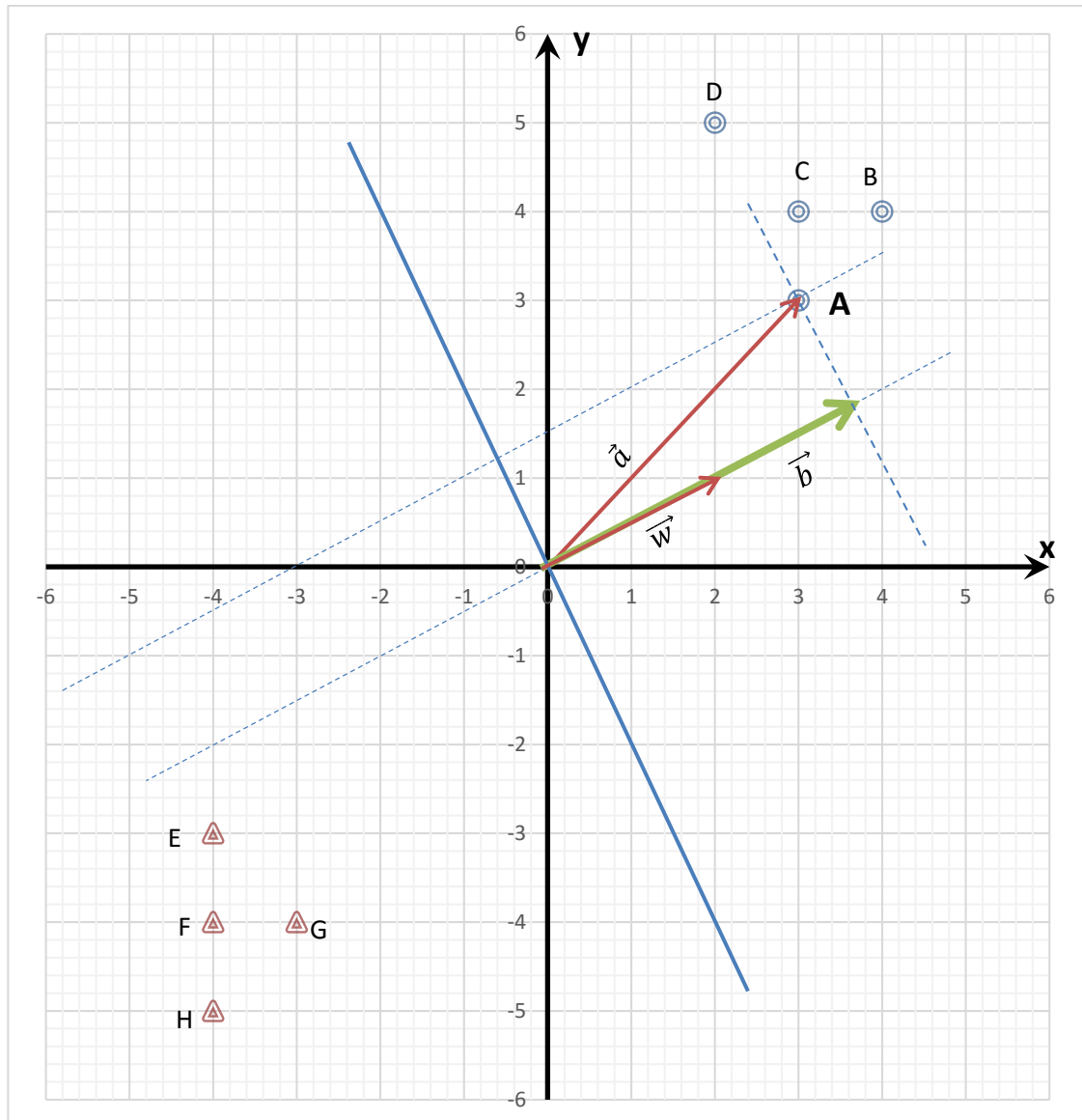


Figure 5: Example dataset 2.

3.3 Finding the Optimal Hyperplane

The calculated margin above M1 (denoted by the blue line) definitely not the largest margin separating perfectly the data. The largest margin should be M2 (denoted by the red line) and by tracing a line cross the middle of M2, there should be the optimal hyperplane. (Figure 5)

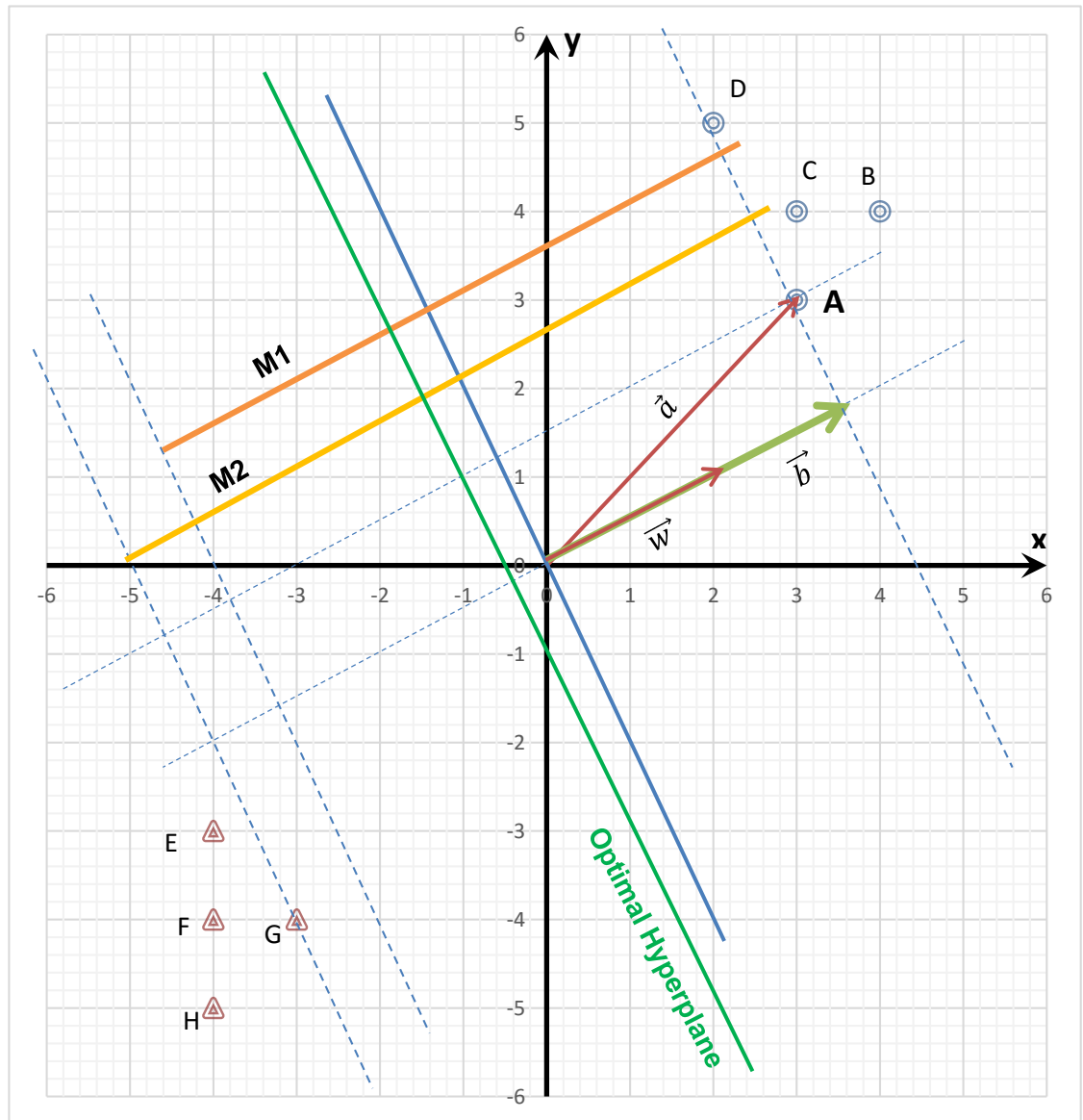


Figure 6: Example dataset 2 with Optimal Hyperplane.

The exact method to find the largest margin can be described in these steps: [15]

- Prepare a dataset
- Choose two hyperplanes so that there are no points between them
- Maximize their distance

Step 1: Gather data set D and classify it

Considering dataset is a collection of n vector \vec{x}_i . Each \vec{x}_i will be associated with y_i indicating if the element belongs to the class (+1) or not (-1). y_i can only have

two possible values: +1 or -1. The dataset can be denoted as the following mathematical notation:

$$D = \{(\vec{x}_i, y_i) \mid x_i \in R^p, y_i \in \{-1, 1\}\}_{i=1}^n$$

In which, R is the real number set, p is the number of dimensions that vector \vec{x}_i has.

Step 2: Select two hyperplanes separating the data with no points between them

For simplicity, assuming that the dataset is linear separatable (can be separated by a line) in a two dimensions space. As mentioned above, the hyperplane equation can be written as: $\vec{w}^T \cdot \vec{x} = 0$, and since we are considering this hyperplane in two dimension, the equation can be simplified as: $\vec{w} \cdot \vec{x} - b = 0$ as the equation of a line. This can be easily proven as below:

Given two 3-dimensional vectors: $\vec{w}^T = (-b, -a, 1)$ and $\vec{x} = (1, x, y)$

$$\vec{w}^T \cdot \vec{x} = -b \times 1 + (-a) \times x + 1 \times y$$

$$\vec{w}^T \cdot \vec{x} = y - ax - b$$

Given two 2-dimensional vectors: $\vec{w}' = (-a, 1)$ and $\vec{x}' = (x, y)$

$$\vec{w}' \cdot \vec{x}' = -a \times x + 1 \times y$$

$$\vec{w}' \cdot \vec{x}' = y - ax$$

Subtract b from both side:

$$\vec{w}' \cdot \vec{x}' - b = y - ax - b$$

Replace $y - ax - b = \vec{w}^T \cdot \vec{x}$:

$$\vec{w}' \cdot \vec{x}' - b = \vec{w}^T \cdot \vec{x}$$

Equation 3: Transformation of hyperplane equation in two dimension

In short $\vec{w} \cdot \vec{x} - b = 0$ can be used to notate the equation of the hyperplane in two dimensions.

Given a hyperplane H_0 separating the dataset and satisfying: $\vec{w} \cdot \vec{x} - b = 0$

Two others hyperplanes H_1 and H_2 can be selected, which also separate the data and have the following equations :

$$\vec{w} \cdot \vec{x} - b = 1$$

And

$$\vec{w} \cdot \vec{x} - b = -1$$

so that H_0 is equidistant from H_1 and H_2 .

Considering the fact that there must be no point between H_1 and H_2 , they must satisfy these following constraint:

For each vector \vec{x}_i either:

$$\vec{w} \cdot \vec{x}_i - b \geq 1 \text{ for } \vec{x}_i \text{ having the class } 1 \text{ or } y_i = 1$$

Or

$$\vec{w} \cdot \vec{x}_i - b \leq -1 \text{ for } \vec{x}_i \text{ having the class } -1 \text{ or } y_i = -1$$

These constraints can be visualized from the Figures 7,8 and 9:

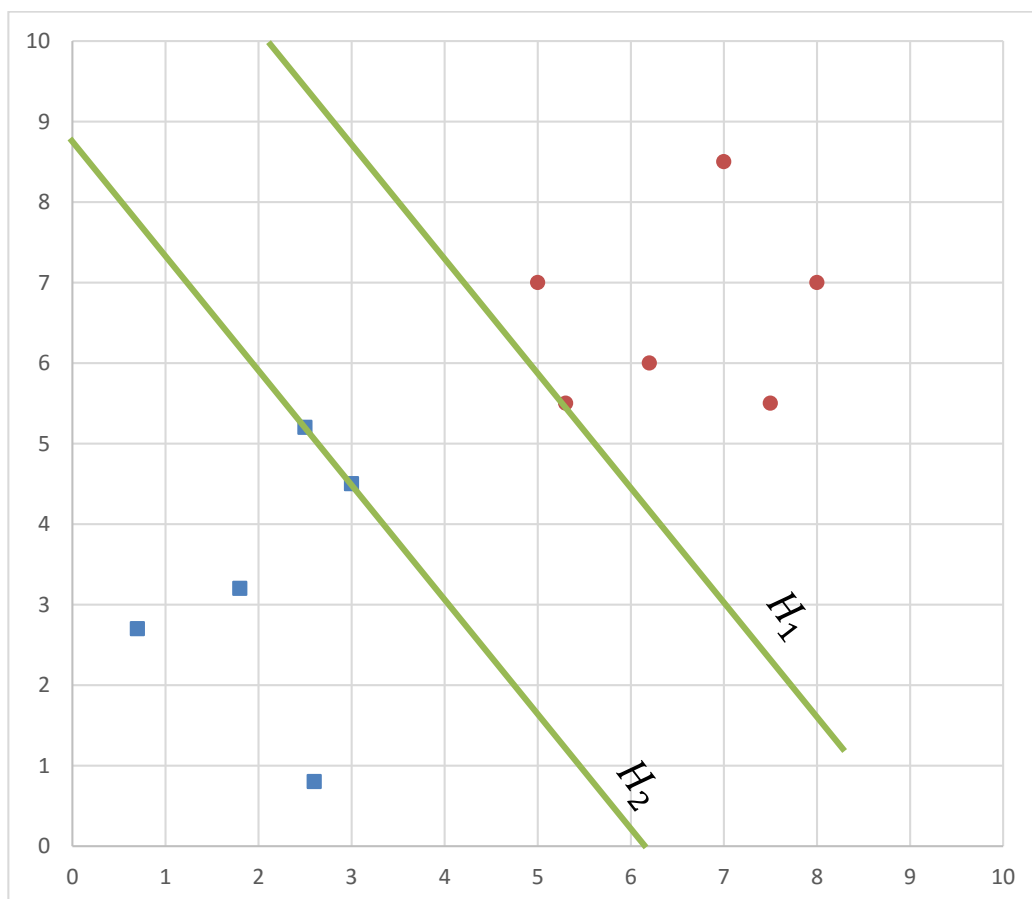


Figure 7: Two Hyperplanes satisfying the constraints.

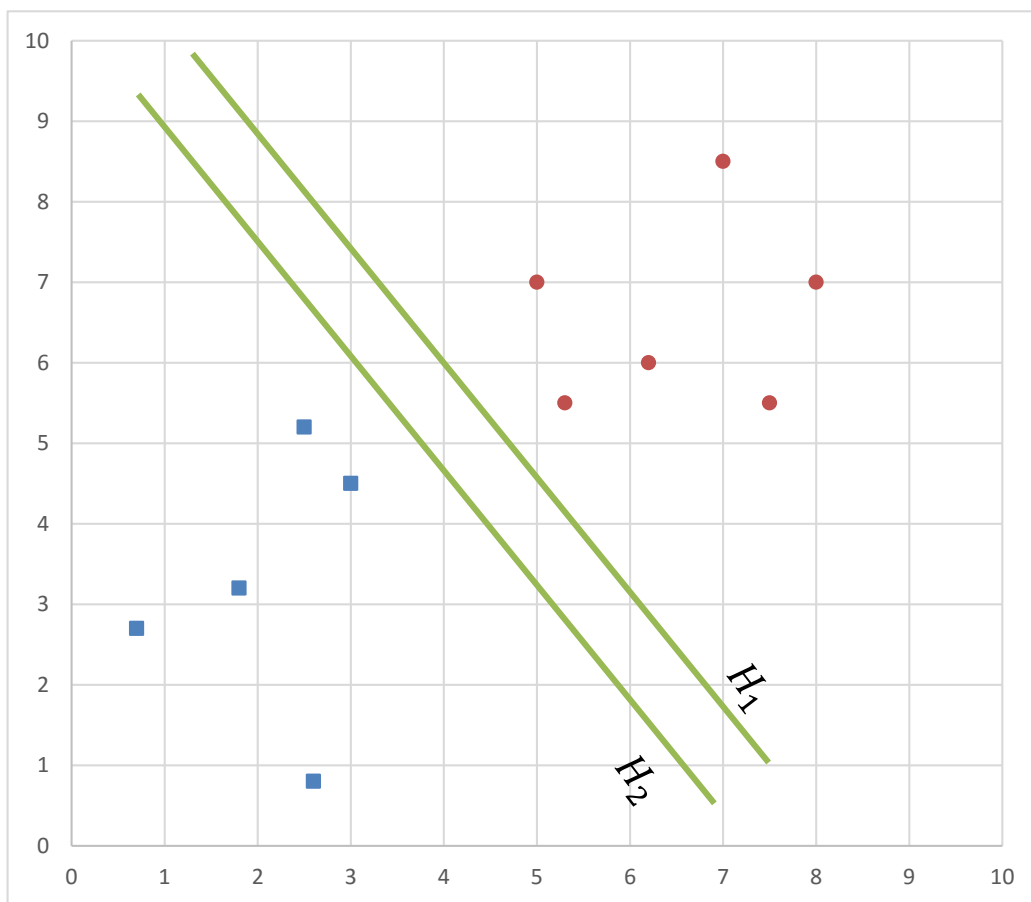


Figure 8: Two Hyperplanes also satisfying the constraints.

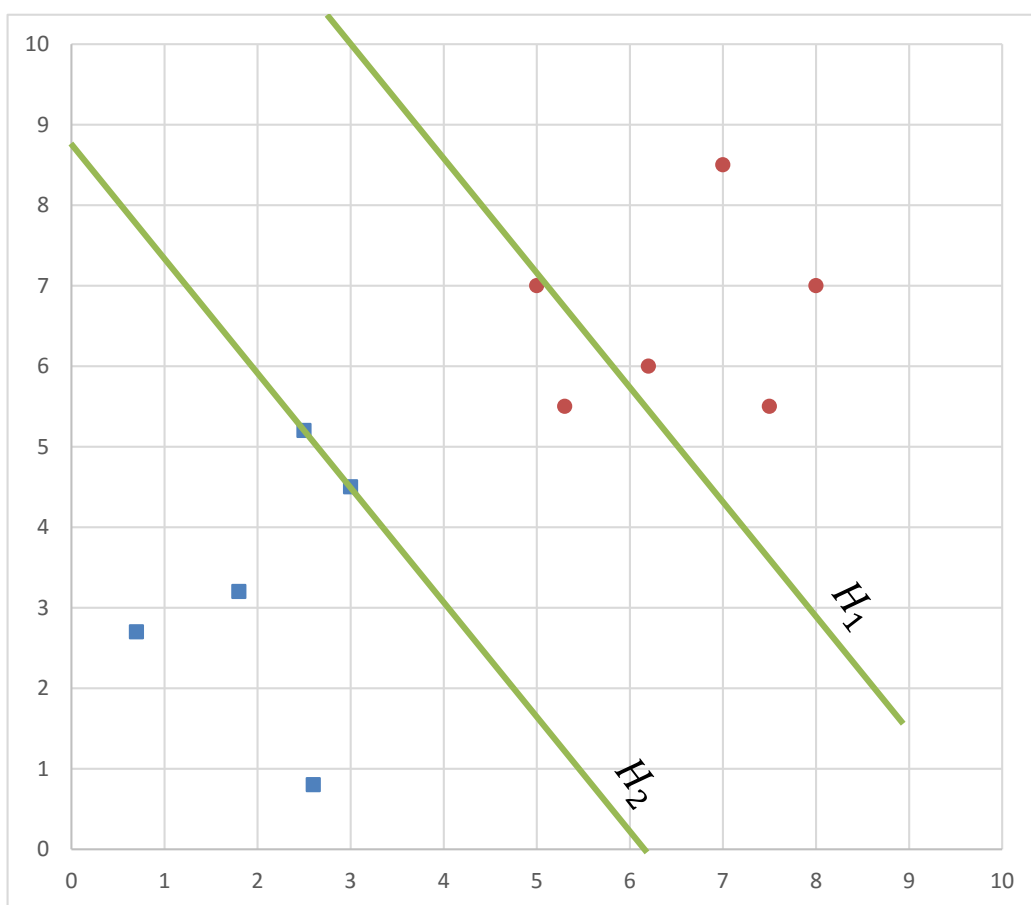


Figure 9: H_1 does not satisfy the first constraint.

One step further, both of the constraints can be combined into one.

Starting with

$$\vec{w} \cdot \vec{x}_i - b \geq 1 \text{ for } \vec{x}_i \text{ having the class 1 or } y_i = 1$$

Multiply both side by $y_i = 1$

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq y_i \Leftrightarrow y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1$$

Continue with

$$\vec{w} \cdot \vec{x}_i - b \leq -1 \text{ for } \vec{x}_i \text{ having the class } -1 \text{ or } y_i = -1$$

Multiply both side by $y_i = -1$

$$\begin{aligned} y_i(\vec{w} \cdot \vec{x}_i - b) &\geq y_i(-1) \Leftrightarrow y_i(\vec{w} \cdot \vec{x}_i - b) \geq (-1) \times (-1) \\ &\Leftrightarrow y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1 \end{aligned}$$

So the two constraints can be combine into one:

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1 \forall 1 \leq i \leq n$$

Equation 4: Combination of the constraints

Step 3: Maximize the distance between two hyperplanes

Let \vec{x}_2 be a point in the hyperplane H_2 and m is the perpendicular distance from \vec{x}_2 to the hyperplane H_1 (by definition, m is the **margin**)

Let \vec{x}_1 be a point in the hyperplane H_1 so that m is the scalar product between \vec{x}_1 and \vec{x}_2

To find \vec{x}_1 , let \vec{k} be the vector whose magnitude is m and perpendicular to H_1

$$\text{Then } \vec{x}_1 = \vec{x}_2 + \vec{k}$$

Let $\vec{u} = \frac{\vec{w}}{|\vec{w}|}$ be the unit vector of \vec{w} (which is the normal vector to the hyperplane H_2). As it is a unit vector, $|\vec{u}| = 1$ and \vec{u} has the same direction as \vec{w} which is perpendicular to H_2

So $\vec{k} = m \vec{u} = m \frac{\vec{w}}{|\vec{w}|}$ will satisfy all the condition

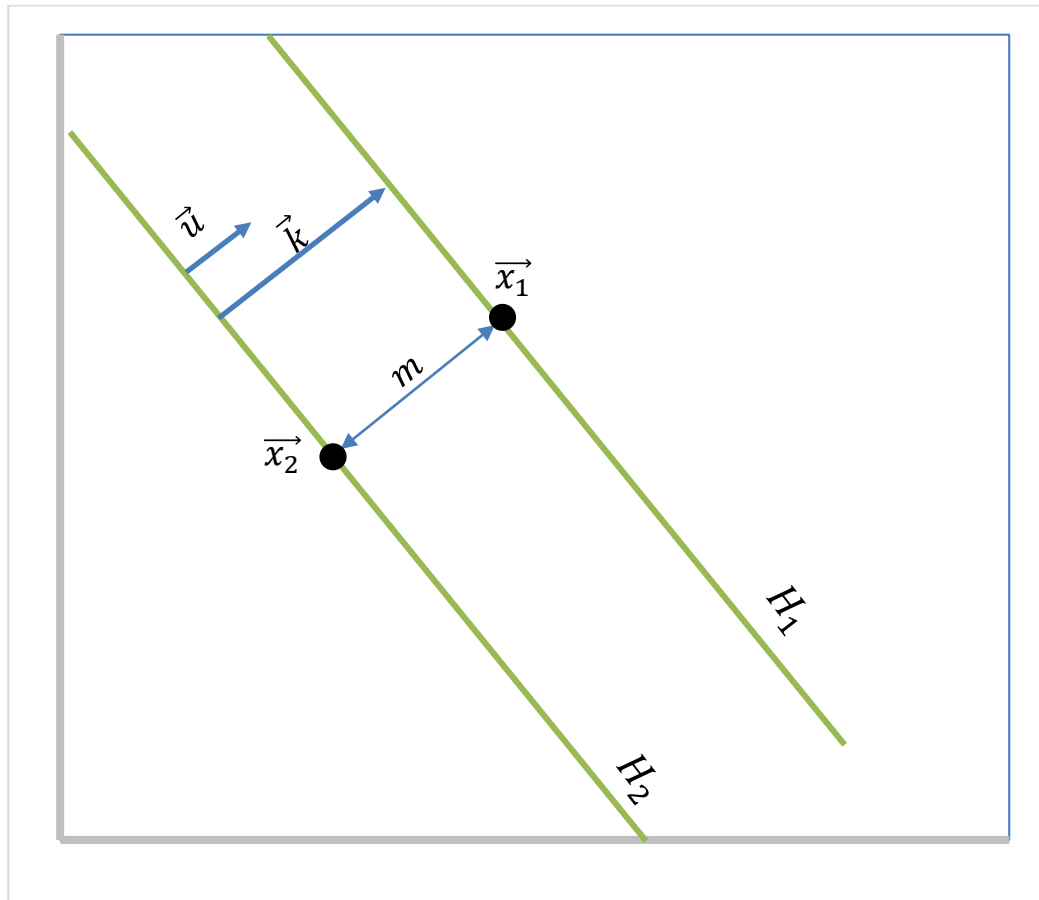


Figure 10: Finding margin m

By definition, \vec{x}_1 is a point in the hyperplane H_1 , so:

$$\vec{w} \cdot \vec{x}_1 - b = 1$$

Replace $\vec{x}_1 = \vec{x}_2 + \vec{k}$

$$\vec{w} \cdot (\vec{x}_2 + \vec{k}) - b = 1$$

Replace $\vec{k} = m \frac{\vec{w}}{|\vec{w}|}$

$$\vec{w} \cdot \left(\vec{x}_2 + m \frac{\vec{w}}{|\vec{w}|} \right) - b = 1$$

$$\vec{w} \cdot \vec{x}_2 + \vec{w} \cdot \frac{\vec{w}}{|\vec{w}|} m - b = 1$$

$$(\vec{w} \cdot \vec{x}_2 - b) + m \frac{\vec{w}^2}{|\vec{w}|} = 1$$

Since $\vec{w} \cdot \vec{x}_1 - b = -1$ and $\vec{w}^2 = |\vec{w}|^2$

$$-1 + m \frac{|\vec{w}|^2}{|\vec{w}|} = -1$$

$$m |\vec{w}| = 2$$

$$m = \frac{2}{|\vec{w}|}$$

Equation 5: Calculation of the margin

In order to maximize m which is the margin, $\frac{2}{|\vec{w}|}$ must be maximized which means $|\vec{w}|$ must be minimized.

After understanding the mechanism behind SVM, in order to build a machine learning program from scratch, some open source tools and libraries has been used which will be mention in the next chapter.

4 OPENCV AND SCIKIT LEARN

OpenCV and Scikit Learn are two free, open source libraries. They are both easy to install and use which makes them perfect for fast prototyping and developing small programs.

4.1 OpenCV library

OpenCV introduction

OpenCV is an open source library released under the BSD license and hence it is free. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 9 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics. [16]

OpenCV library can be applied in many area including:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality

OpenCV also includes a statistical machine learning library such as:

- k-nearest neighbor algorithm
- Naive Bayes classifier

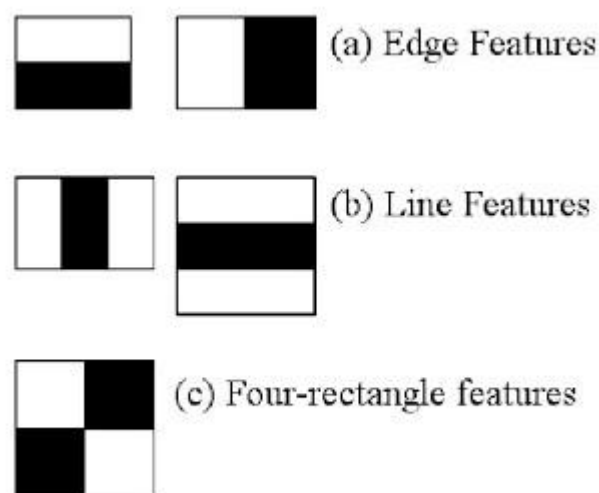
- Artificial neural networks
- Random forest
- Support vector machine (SVM)

OpenCV has C++, C, JAVA, Matlab and Python interfaces, supports multiple OS (Windows, Linux, Mac OS). OpenCV leans mostly towards real-time vision applications and takes advantages of MMX (instruction set designed by Intel) and SSE (**Streaming SIMD Extensions** - instruction set extension to the x86 architecture) when available.

If the library finds Intel's Integrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate itself.

OpenCV Face Detection using Haar Cascades

Object Detection using Haar feature is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. Initially, the algorithm needs a lot of images with faces and images without faces to train the classifier. Then features need to be extracted from the classifier. In order to achieve this, Haar features as shown in Picture 4 are used.



Picture 4: Haar features. Source: <http://opencv-python-tutroals.readthedocs.io/>

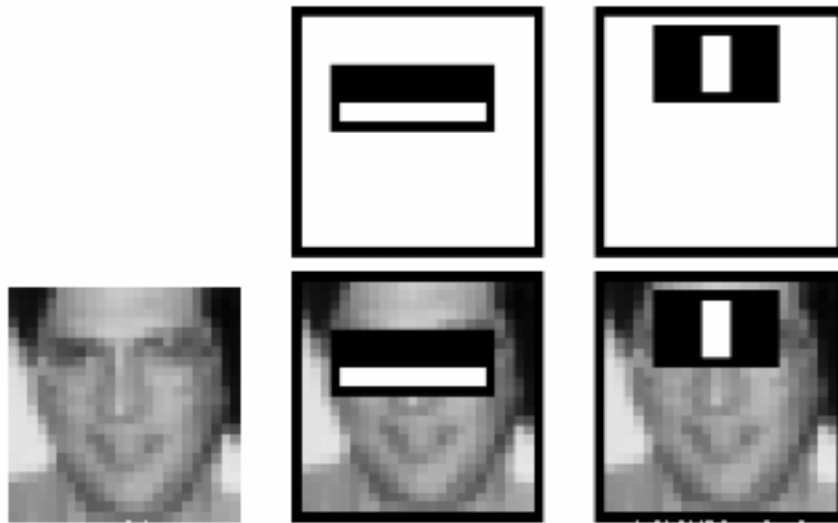
Each feature is a single value obtained by subtracting sum of pixels under white rectangle from the sum of pixels under black rectangle. Rectangle features can be computed very rapidly using an intermediate representation for the image

which called the integral image. It simplifies calculation of sum of pixels, no matter how large may be the number of pixels, to an operation involving just four pixels. It speeds up dramatically the calculation process. There is a problem among all the features that calculated using Haar features, that is most of the features are irrelevant. For instant, consider Picture 5. Top row shows two good features. The first feature can be used to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature also can be used to focus on the same property (the eyes are darker than the bridge of the nose). The same windows cannot be applied on cheeks or any other place. So, selecting the best features out of thousands other features can be achieved by Adaboos (Adaptive Boosting).

The first step is to apply each and every feature on all the training images. For each feature, it will find the best threshold to classify if the faces present or not. There will be errors or misclassifications. By selecting the features with minimum error rate, which means they are the features that best classifies faces. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. New error rates are calculated and so as new error rates. This loop will continue until required accuracy or error rate is achieved or required number of features are found. Final classifier is created by combining all of the weak classifiers (they are called weak classifiers because them alone can not classify the image but together they form a strong classifier).

In an image, most of the image region doesn't contain any faces. It is a better idea to have a simple method to check if a window is not a face region, discard and don't process it. This can speed up the process of face recognition by focusing on the face region only instead of all areas. The method here is using the concept of Cascade of Classifiers. Instead of applying all training features, group the features into different stages of classifiers and apply one-by-one. If a window fails the first stage, it will be discarded immediately, if it passes, apply the second stage of features, keep continuing the loop. Finally, the window which

passes all stages is a face region. This explains how Face Detection using Haar Cascade works.



Picture 5: Haar features in face recognition. Source: <http://opencv-python-tutroals.readthedocs.io/>

4.2 Sci-kit Learn

Scikit Learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. The library is built upon the SciPy (Scientific Python) that must be installed before you can use Scikit Learn. This stack that includes:

- NumPy: Base n-dimensional array package
- SciPy: Fundamental library for scientific computing
- Matplotlib: Comprehensive 2D/3D plotting
- IPython: Enhanced interactive console
- SymPy: Symbolic mathematics
- Pandas: Data structures and analysis

Extensions or modules for SciPy are conventionally named SciKits. As such, the module provides learning algorithms and is named Scikit Learn. The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as ease of use, code quality, collaboration, documentation and performance.

Some popular groups of models provided by Scikit Learn include:

- Clustering: for grouping unlabeled data such as KMeans.
- Cross Validation: for estimating the performance of supervised models on unseen data.
- Datasets: for test datasets and for generating datasets with specific properties for investigating model behavior.
- Dimensionality Reduction: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
- Ensemble methods: for combining the predictions of multiple supervised models.
- Feature extraction: for defining attributes in image and text data.
- Feature selection: for identifying meaningful attributes from which to create supervised models.
- Parameter Tuning: for getting the most out of supervised models.
- Manifold Learning: For summarizing and depicting complex multi-dimensional data.
- Supervised Models: a vast array not limited to generalized linear models, but also including discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

5 THE SMILE RECOGNITION PROGRAM

With the purpose of demonstrate the capability of Machine Learning, a program has been written in Python using Scikit Learn and OpenCV library. The program will first be trained from the built-in dataset from Scikit Learn and then using OpenCV, the program will take live input from a webcam and make the decision whether the input (a face video) is smiling or not, **in real time**.

5.1 The preparation of the dataset

The data set used in this program is the Olivetti faces dataset which contains a set of face images taken between April 1992 and April 1994 at AT&T Laboratories Cambridge. As described on the original website:

There are ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement).

The images were quantized to 256 grey levels and stored as unsigned 8-bit integers; the loader would convert these to floating point values on the interval [0, 1], which are easier to work with for many algorithms.

The “target” for this database was an integer from 0 to 39 indicating the identity of the person pictured; however, with only 10 examples per class, this relatively small dataset is more interesting from an unsupervised or semi-supervised perspective.

To prepare this dataset, a simple program has been created, whose function is just take user respond to decide if the face is smile or not and then put it in a library for later use. The code can be found in Appendix 1.

First, the dataset needed to be fetched by line 8. After the dataset was loaded, a simple interface was built to take user input to classify whether the input face was smiling or not. All that is done by using Matplotlib. `class Trainer` contained

3 functions, `__init__` would start the trainer, `increment_face` would have a counter to increment whenever the button press to classify faces and lastly `record_result` would save the result. After that the interface was built by the rest of the code. Finally, the result was saved as a .xml file with the last command:

```
with open('results.xml', 'w') as f:
    json.dump(trainer.results, f)
```

Below is the simple interface of this small program:

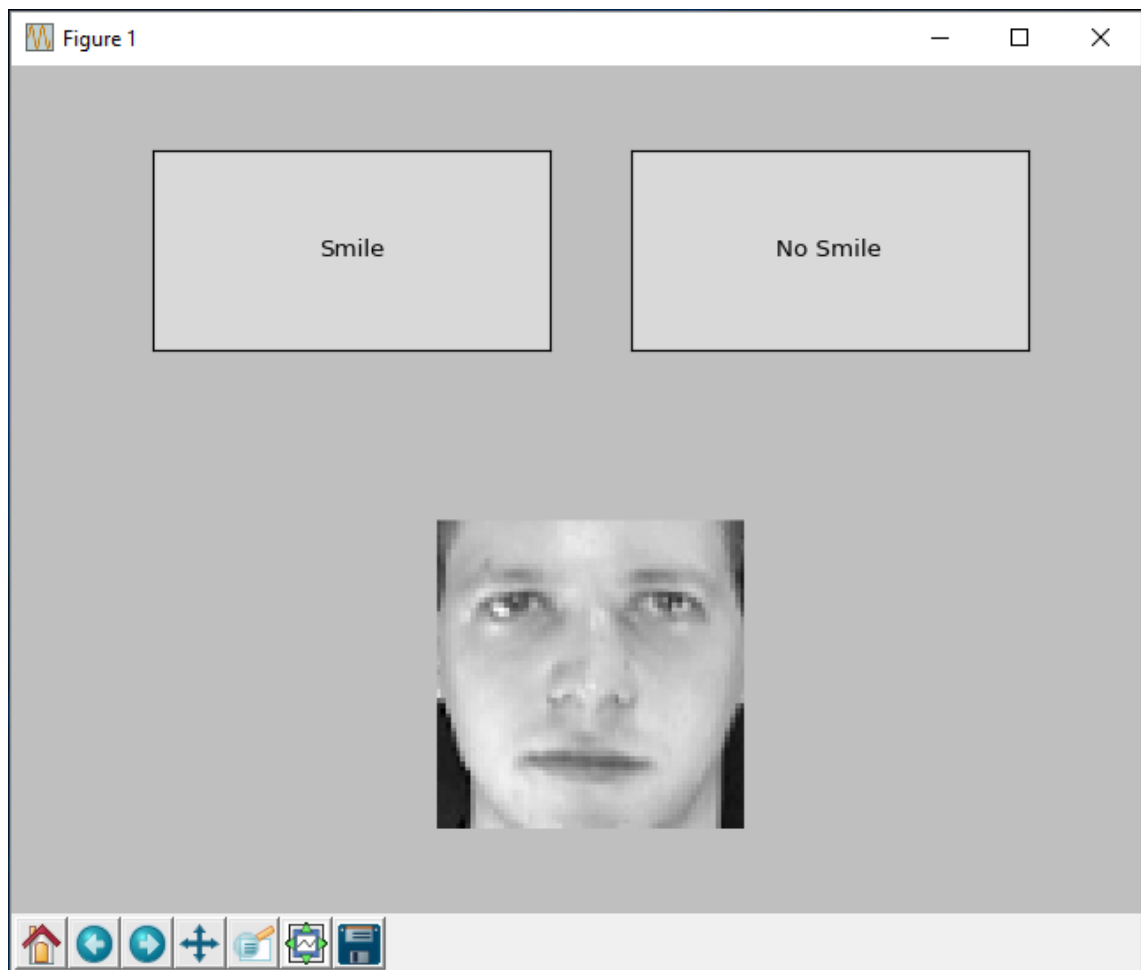


Figure 11: Simple interface classification program

After classified 400 faces and all the results were saved, the statistic of the dataset could be displayed using the code in appendix 2.

Again, this is just a simple program built upon Matplotlib. Line 9 to 15 in the code display the statistic between smile and no smile images:

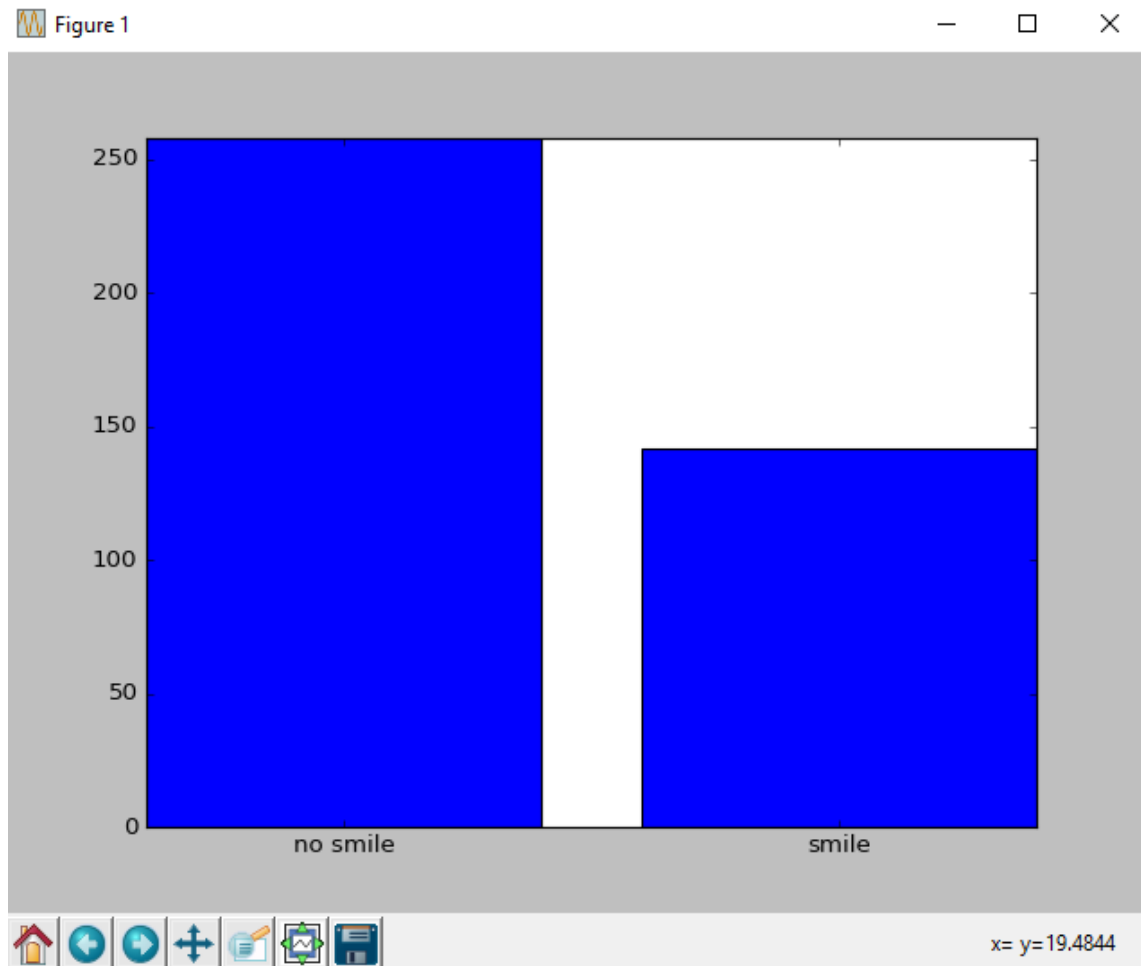


Figure 12: Smile vs. no smile statistic

The rest of the code displays and labels just the smiling and not smiling face just for the purpose of reviewing:



Figure 13: Smiling faces

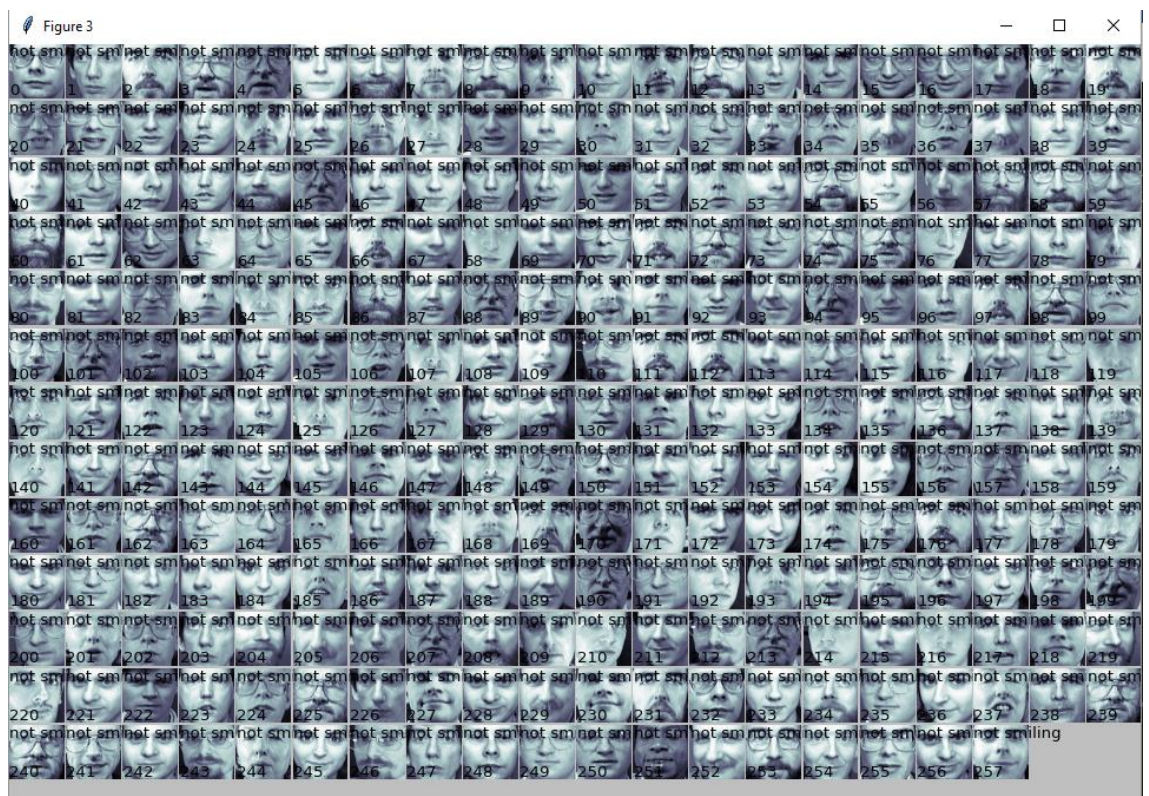


Figure 14: Not smiling faces

5.2 Training the classifier

After classifying all the data the next step is to train the classifier to recognize if a face is smiling or not.

The classifier trainer can be completed by the code in the appendix 3.

First the classifier was initialized in line 12-13. Then, with line 14-15, the dataset was built by combining the face images with the result that had been saved from above. After that, in line 16-17, the target vector determined whether smiling was true (denote by 1) or not smiling was true (denote by 0). The rest of the code dedicated to train the classifier with 5-fold cross validation. [17].

Output:

```
[ 0.85          0.86666667  0.75          0.76666667  0.88333333]
```

```
Mean score: 0.823 (+/-0.027)
```

```
Accuracy on training set:
```

```
1.0
```

```
Accuracy on testing set:
```

```
0.84
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	0.88	0.88	0.88	67
1	0.76	0.76	0.76	33
avg / total	0.84	0.84	0.84	100

```
Confusion Matrix:
```

```
[[59  8]
```

```
 [ 8 25]]
```

```
[Finished in 2.7s]
```

The training was done and the result look promising with accuracy on testing set was around 84%.

In order to verify this, the author took an image of himself, uses it as input and extracted the face region and fed it through the classifier to test if it works or not. It was done by the code in the appendix 4.

In here, first, the cascade and author's image is loaded by line 6-7, after that a gray scale was define to convert the image to gray color, it was done by line 8-9. From line 10 through 17, the face region would be highlighted:

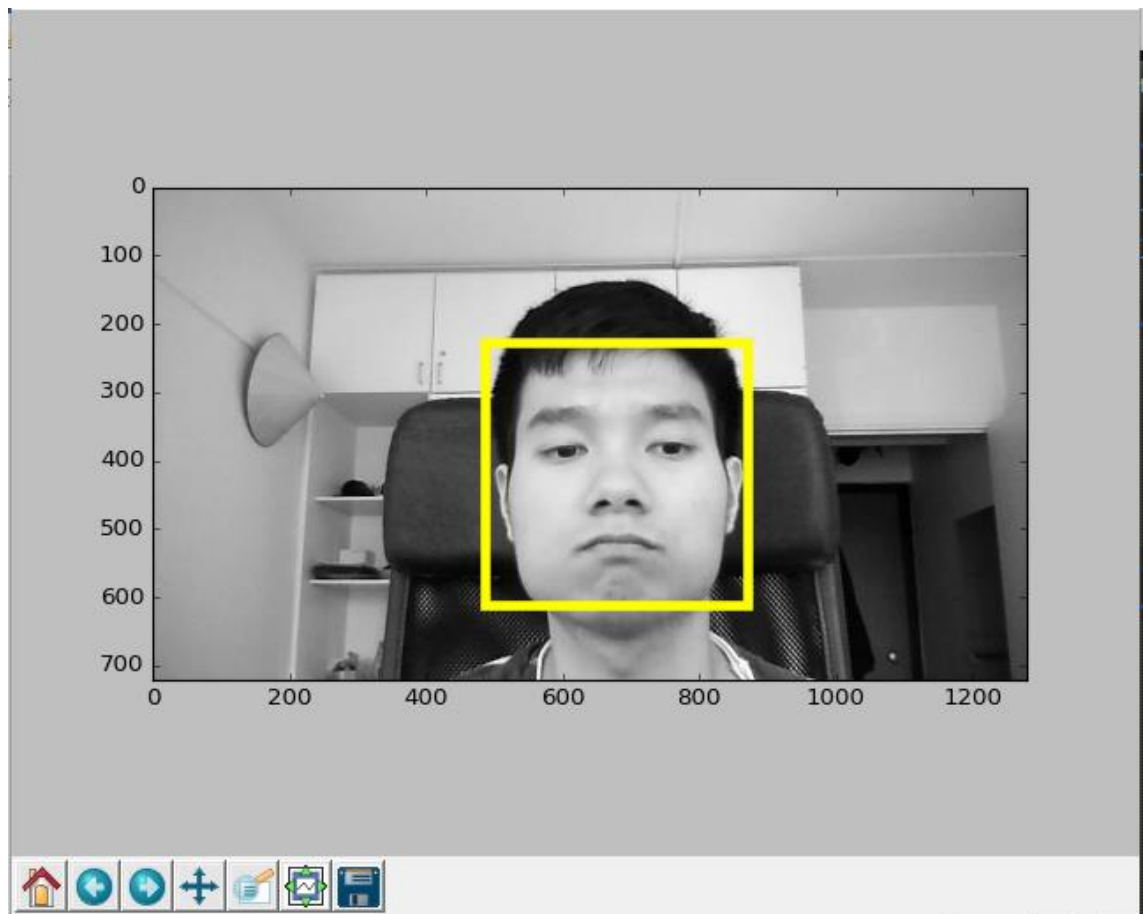


Figure 15: Highlighted face region

Line 18 to 26 would extract just the face region and convert it to the same format as the dataset face images:

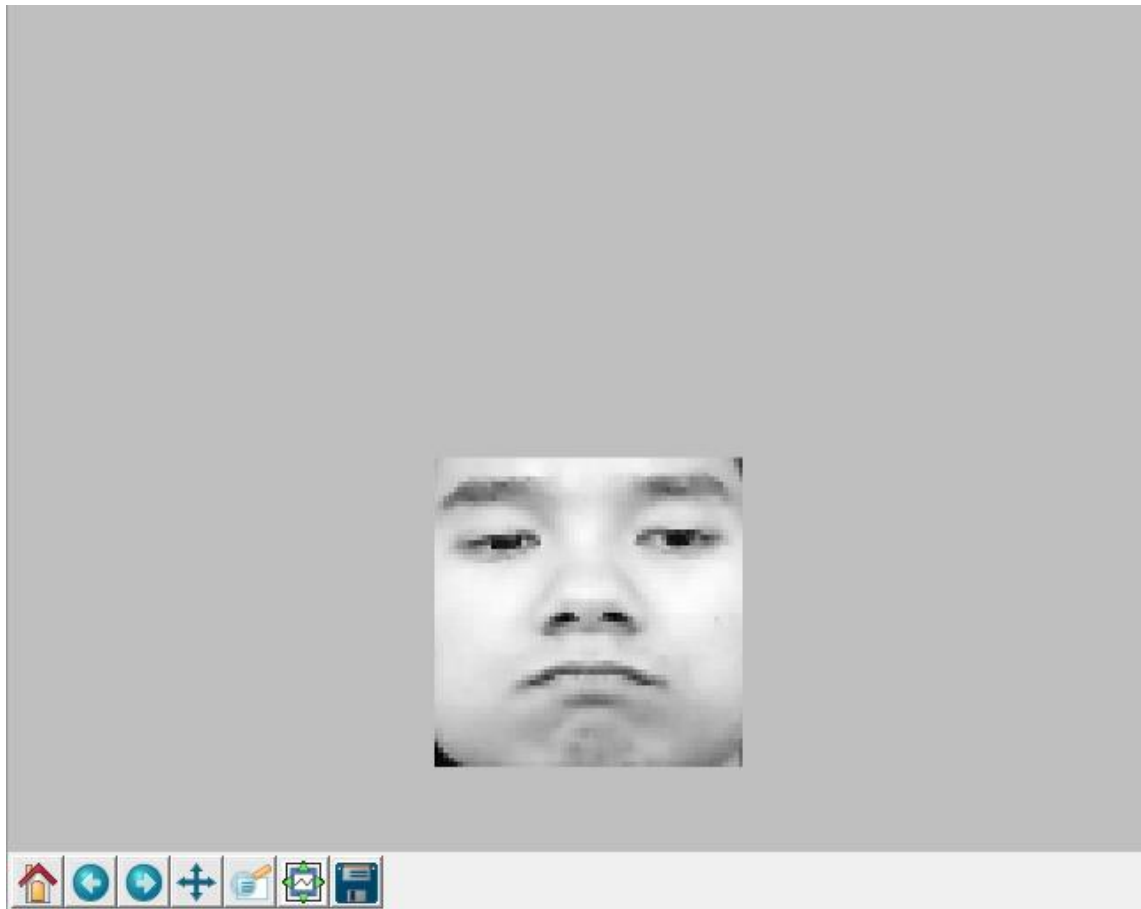


Figure 16: Extracted face

And finally, this extracted image was put through the classifier by the line 27, and the output is:

```
this person is smiling: [0]
```

It means the classifier predicted this face as not smiling, which was indeed true.

5.3 Smile Detection using live webcam

To finish the program, using OpenCV and a webcam, the live video captured from webcam is used and extracted frame by frame then all of the frames are run through the classifier, return result in real-time and display back to the screen, the detailed code can be found in Appendix 5. [18]

`detect_face` was used to detect face appear from the frame by using Haar cascade.

`extract_face_features` was used to extract the face region and convert to the format that the classifier accepts.

After that, the functions were run through a loop in which it took input from the video from the webcam (line 22) and predicted them (line 38), then the result was returned and displayed (line 40 – 43). The output was a real-time smile detection program:

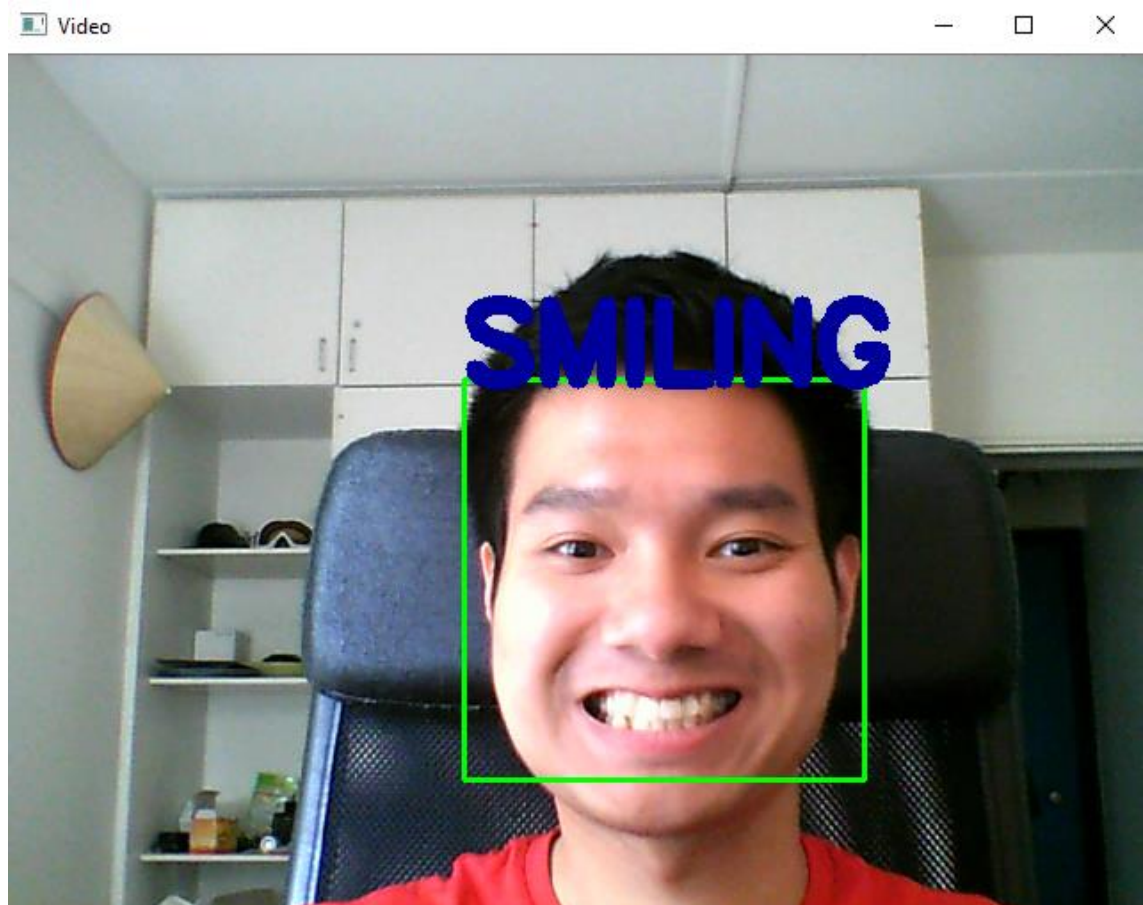


Figure 17: Live detection - smile

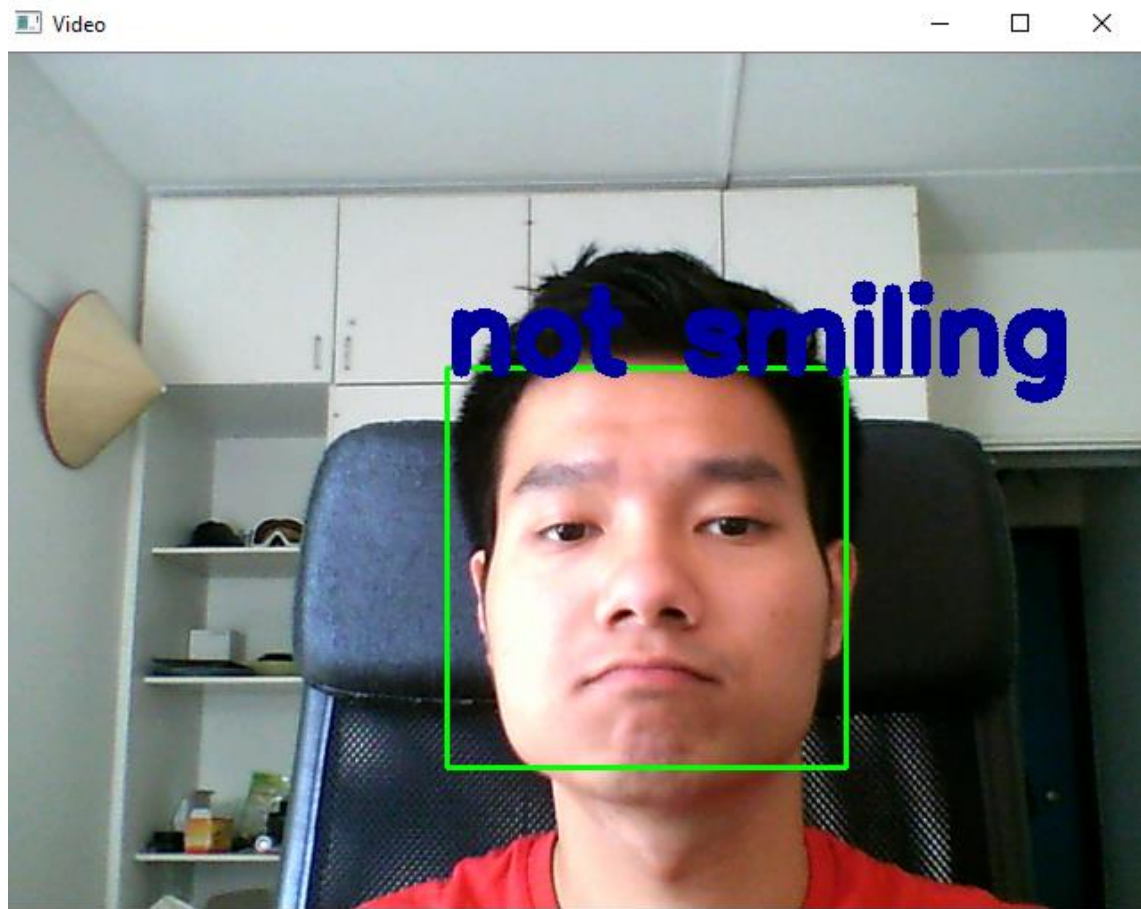


Figure 18: Live detection - no smile

As mentioned above, while training the classifier, the accuracy was around 84%. The program will fail to predict if there would be any sudden change in the input, such as face angle, lighting condition ... This because the dataset's size is relatively small, and it could not cover every single situation in real life. In order to improve the accuracy and performance of the program, more data are needed to feed through the classifier.

6 CONCLUSION

As a developing field, Machine Learning is evolving at a much faster pace than any other field in IT. More algorithms are being developed and also the old algorithms are improved over time. Also applications of Machine Learning are reaching far beyond just solving classification problems. Understanding the basic of Machine Learning and the mechanism of the algorithms will help to make use of this powerful tool in order to apply to many more area in the human society.

This thesis' purpose was to introduce Machine Learning and demonstrate the application by using Machine Learning algorithm, it is possible to train the machine to recognize facial emotion without using hardcoded arguments. The program is not perfect, mainly due to the size of the dataset provided (relatively small) and the error of SVM. To improve the program, more data need to be provided and classified, which will increase accuracy and performance of the program. Other algorithms could also be tested to see which classification methods work best with the dataset provided. The program just represents a tiny fraction in the Machine Learning field, what its capability is.

In the near future, Machine Learning will be one of the most important field in Computer Science. Freeing programs from the traditional hard-code method to move on the new era of Machine Learning can be consider as one of the greatest breakthrough in the IT world. With the ever changing world, adaptive programs not only reduce the workload for human but they also can develop new invention through random mutation code, just like the way human discover the world but much faster.

REFERENCES

- [1] S. Arthur .1958. '*Some studies in machine learning using the game of checkers*', IBM Journal of Research and Development, Volume:44 Issue:1.2
Available from: IEEE Digital Explore Library
- [2] T. Mitchell .1997. '*Machine Learning*' [pdf]
Available at <https://www.cs.swarthmore.edu/~meeden/cs63/f11/ml-intro.pdf>
- [3] P.R. Cohen .2006. '*If not Turing's Test, then what?*' AI Magazine Volume 26 Number 4
Available at: <https://www.aaai.org/ojs/index.php/aimagazine/article/download/1849/1747>
- [4] Y. Freund, R. E. Schapire. .1999. '*Large margin classification using the perceptron algorithm*' [pdf]
Available at: <http://cseweb.ucsd.edu/~yfreund/papers/LargeMarginsUsingPerceptron.pdf>
- [5] J. McCarthy and E. Feigenbaum .1990. '*In Memoriam Arthur Samuel: Pioneer in Machine Learning*', AI Magazine Volume 11 Number 3
Available at: <http://www.aaai.org/ojs/index.php/aimagazine/article/view/840/758>
- [6] T.M. Cover and P.E. Hart .1967. '*Nearest neighbor pattern classification*', IEEE Transactions on Information Theory, Volume:13 Issue:1
Available from: IEEE Digital Explore Library
- [7] L. Earnest .2012. '*Stanford Cart*'
Available at: <http://web.stanford.edu/~learnest/cart.htm>
- [8] NYtimes .August 16, 1988. '*Learning, Then Talking*'
Available at: <http://www.nytimes.com/1988/08/16/science/learning-then-talking.html>
- [9] C. Krauthammer .May 26, 1997. '*Be Afraid*', THE MAGAZINE: From the May 26 Issue
Available at: <http://www.weeklystandard.com/be-afraid/article/9802>
- [10] G.E. Hinton .October 2007. '*Learning multiple layers of representation*' ,TRENDS in Cognitive Sciences Vol.11 No.10
Available at: <http://www.cs.toronto.edu/~fritz/absps/tics.pdf>
- [11] M. Mohri, A. Rostamizadeh and A. Talwalkar .2012. '*Foundations of Machine Learning*'
Available from: <https://openlibrary.org/> ISBN 9780262018258.
- [12] G.M. James .2003. '*Variance and bias for general loss functions*' [pdf]
Available at: <http://www-bcf.usc.edu/~gareth/research/bv.pdf>
- [13] P.M. Baggenstoss .Jan. 2004. '*Class-specific classifier: avoiding the curse of dimensionality*', IEEE Aerospace and Electronic Systems Magazine, Volume:19 Issue:1
Available from: IEEE Digital Explore Library
- [14] C.E. Brodely and M.A. Friedl, .1999. '*Identifying and Eliminating Mislabeled Training Instances*', Journal of Artificial Intelligence Research 11

Available at: <http://jair.org/media/606/live-606-1803-jair.pdf>

[15] C.D. Manning, P. Raghavan and H. Schütze .2008. *'Introduction to Information Retrieval'*, , Cambridge University Press

Available at: <http://nlp.stanford.edu/IR-book/html/htmledition/irbook.html>

[16] About OpenCV

Available at: <http://opencv.org/about.html>

[17] Supervised Learning - Image Recognition with Support Vector Machines

Available at: <http://nbviewer.jupyter.org/github/gmonce/scikit-learn-book/blob/master/Chapter%20-%20Supervised%20Learning%20-%20Image%20Recognition%20with%20Support%20Vector%20Machines.ipynb>

[18] Face Detection in Python Using a Webcam

Available at: <https://realpython.com/blog/python/face-detection-in-python-using-a-webcam/>

Appendix 1. Simple program to take user respond for classifying data

```

1. import matplotlib
2. import matplotlib.pyplot as plt
3. import numpy as np
4. from sklearn import datasets
5. from matplotlib.widgets import Button
6. import json
7. fig = plt.figure()
8. faces = datasets.fetch_olivetti_faces()
9. class Trainer:
10.     def __init__(self):
11.         a. self.results = {}
12.         b. self.imgs = faces.images
13.         c. self.index = 0
14.     def increment_face(self):
15.         a. if self.index + 1 >= len(self.imgs):
16.             i. return self.index
17.         b. else:
18.             i. while str(self.index) in self.results:
19.                 ii. print(self.index)
20.                 iii. self.index += 1
21.             iv. return self.index
22.     def record_result(self, smile=True):
23.         a. self.results[str(self.index)] = smile
24.         trainer = Trainer()
25.     def display_face(face):
26.         axim = fig.add_subplot(2,1,2)
27.         axim.imshow(face, cmap='gray', interpolation='none', )
28.         axim.axis("off")
29.
30.     def update_smile(b):
31.         trainer.record_result(smile=True)
32.         trainer.increment_face()
33.         display_face(trainer.imgs[trainer.index])
34.
35.     def update_no_smile(b):
36.         trainer.record_result(smile=False)
37.         trainer.increment_face()
38.         display_face(trainer.imgs[trainer.index])
39.
40.         axSmile = fig.add_subplot(3,2,1)
41.         axNosmile = fig.add_subplot(3,2,2)
42.         bSmile = Button(axSmile, 'Smile')
43.         bSmile.on_clicked(update_smile)
44.         bNosmile = Button(axNosmile, 'No Smile')
45.         bNosmile.on_clicked(update_no_smile)
46.         display_face(trainer.imgs[trainer.index])
47.         plt.show()
48.         with open('results.xml', 'w') as f:
49.             json.dump(trainer.results, f)

```

Appendix 2. Display the statistic of the dataset

```

1. import matplotlib.pyplot as plt
2. from sklearn.svm import SVC
3. from sklearn.cross_validation import train_test_split
4. from sklearn.cross_validation import cross_val_score, KFold
5. import json
6. from sklearn import datasets

7. faces = datasets.fetch_olivetti_faces()
8. results = json.load(open('results.xml'))

9. yes, no = (sum([results[x] == True for x in results]),
             i. sum([results[x] == False for x in results]))
10. plt.bar([0, 1], [no, yes])
11. plt.ylim(0, max(no, yes))
12. plt.xticks([0.4, 1.4], ['no smile', 'smile']);
13. smiling_indices = [int(i) for i in results if results[i] ==
    True]
14. fig = plt.figure(figsize=(12, 12))
15. fig.subplots_adjust(left=0, right=1, bottom=0, top=1,
    hspace=0.05, wspace=0.05)
16. for i in range(len(smiling_indices)):
17.     # plot the images in a matrix of 20x20
18.     p = fig.add_subplot(20, 20, i + 1)
19.     p.imshow(faces.images[smiling_indices[i]], cmap=plt.cm.bone)
20.     # label the image with the target value
21.     p.text(0, 14, "smiling")
22.     p.text(0, 60, str(i))
23.     p.axis('off')
24. no_smiling_indices = [int(i) for i in results if results[i] ==
    False]
25. fig = plt.figure(figsize=(12, 12))
26. fig.subplots_adjust(left=0, right=1, bottom=0, top=1,
    hspace=0.05, wspace=0.05)
27. for i in range(len(no_smiling_indices)):
28.     # plot the images in a matrix of 20x20
29.     p = fig.add_subplot(20, 20, i + 1)
30.     p.imshow(faces.images[no_smiling_indices[i]], cmap=plt.cm.bone)
31.     # label the image with the target value
32.     p.text(0, 14, "not smiling")
33.     p.text(0, 60, str(i))
34.     p.axis('off')
35. plt.show()

```

Appendix 3. Classifier trainer

```

1. import matplotlib
2. import matplotlib.pyplot as plt
3. from sklearn.svm import SVC
4. from sklearn.cross_validation import train_test_split
5. from sklearn.cross_validation import cross_val_score, KFold
6. import json
7. from sklearn import datasets, metrics
8. from scipy.stats import sem
9. import numpy as np

10. faces = datasets.fetch_olivetti_faces()
11. results = json.load(open('results.xml'))
12. from sklearn.svm import SVC
13. svc_1 = SVC(kernel='linear')
14. indices = [i for i in results]
15. data = faces.data[indices, :]
16. target = [results[i] for i in results]
17. target = np.asarray(target).astype(int)
18. X_train, X_test, y_train, y_test = train_test_split(
    a. data, target, test_size=0.25, random_state=0)
19. def evaluate_cross_validation(clf, X, y, K):
20.     # create a k-fold cross validation iterator
21.     cv = KFold(len(y), K, shuffle=True, random_state=0)
22.     # by default the score used is the one returned by score method
    of the estimator (accuracy)
23.     scores = cross_val_score(clf, X, y, cv=cv)
24.     print (scores)
25.     print ("Mean score: {0:.3f} (+/-
    {1:.3f})".format(np.mean(scores), sem(scores)))
26.     evaluate_cross_validation(svc_1, X_train, y_train, 5)
27. def train_and_evaluate(clf, X_train, X_test, y_train, y_test):
28.     clf.fit(X_train, y_train)
29.     print ("Accuracy on training set:")
30.     print (clf.score(X_train, y_train))
31.     print ("Accuracy on testing set:")
32.     print (clf.score(X_test, y_test))

33. y_pred = clf.predict(X_test)

34. print ("Classification Report:")
35. print (metrics.classification_report(y_test, y_pred))
36. print ("Confusion Matrix:")
37. print (metrics.confusion_matrix(y_test, y_pred))

38. train_and_evaluate(svc_1, X_train, X_test, y_train, y_test)

```


Appendix 4. Testing the classifier

```

1. import cv2
2. from matplotlib.patches import Rectangle
3. import matplotlib.pyplot as plt
4. import numpy as np
5. from scipy.ndimage import zoom

6. face_cascade =
   cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
7. img = cv2.imread('mah_face3.jpg')
8. gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
9. faces = face_cascade.detectMultiScale(gray, 1.3, 5)
10. ax = plt.gca()
11. ax.imshow(gray, cmap='gray')
12. for (x,y,w,h) in faces:
13.     ax.add_artist(Rectangle((x, y), w, h, fill=False, lw=5,
        color='yellow'))
14.     original_extracted_face = gray[y:y+h, x:x+w]
15.     horizontal_offset = 0.15 * w
16.     vertical_offset = 0.2 * h
17.     extracted_face = gray[y+vertical_offset:y+h,
        a. x+horizontal_offset:x+horizontal_offset+w]
18.     new_extracted_face = zoom(extracted_face, (64. /
        extracted_face.shape[0],
        a. 64. /
        extracted_face.shape[1]))
19.     new_extracted_face = new_extracted_face.astype(float)
20.     new_extracted_face /= float(new_extracted_face.max())
21.     fig = plt.figure()
22.     def display_face(face):
23.         axim = fig.add_subplot(2,1,2)
24.         axim.imshow(face, cmap='gray', interpolation='none', )
25.         axim.axis("off")
26.         display_face(new_extracted_face[:, :])
27.         print("this person is smiling:
        {0}".format(svc_1.predict(new_extracted_face.ravel()))
28.         plt.show()

```

Appendix 5. Running classifier through video loop

```

1. def detect_face(frame):
2.     cascPath = "haarcascade_frontalface_default.xml"
3.     faceCascade = cv2.CascadeClassifier(cascPath)
4.     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
5.     detected_faces = faceCascade.detectMultiScale(gray, 1.3, 5)
6.     return gray, detected_faces
7. def extract_face_features(gray, detected_face, offset_coefficients):
8.     (x, y, w, h) = detected_face
9.     horizontal_offset = offset_coefficients[0] * w
10.    vertical_offset = offset_coefficients[1] * h
11.    extracted_face = gray[y+vertical_offset:y+h,
12.    x+horizontal_offset:x+horizontal_offset+w]
13.    new_extracted_face = zoom(extracted_face, (64. /
14.    extracted_face.shape[0],
15.    64. / extracted_face.shape[1]))
16.    new_extracted_face = new_extracted_face.astype(float)
17.    new_extracted_face /= float(new_extracted_face.max())
18.    return new_extracted_face
19.    def predict_face_is_smiling(extracted_face):
20.    return svc_1.predict(extracted_face.ravel())

20.     cascPath = "haarcascade_frontalface_default.xml"
21.     faceCascade = cv2.CascadeClassifier(cascPath)
22.     video_capture = cv2.VideoCapture(0)

23.     while True:
24.         # Capture frame-by-frame
25.         ret, frame = video_capture.read()
26.         # detect faces
27.         gray, detected_faces = detect_face(frame)
28.         face_index = 0
29.         # predict output
30.         for face in detected_faces:
31.             (x, y, w, h) = face
32.             if w > 100:
33.                 # draw rectangle around face
34.                 cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
35.                 # extract features
36.                 extracted_face = extract_face_features(gray, face, (0.03, 0.05))
37.                 # predict smile
38.                 prediction_result = predict_face_is_smiling(extracted_face)
39.                 # annotate main image with a label
40.                 if prediction_result == 1:
41.                     cv2.putText(frame, "SMILING", (x,y), cv2.FONT_HERSHEY_SIMPLEX, 2,
155, 10)
42.                 else:
43.                     cv2.putText(frame, "not smiling", (x,y),
44.                     cv2.FONT_HERSHEY_SIMPLEX, 2, 155, 10)
45.                 # increment counter
46.                 face_index += 1
47.                 # Display the resulting frame
48.                 cv2.imshow('Video', frame)
49.                 if cv2.waitKey(1) & 0xFF == ord('q'):
50.                     break
51.                 # When everything is done, release the capture
52.                 video_capture.release()
53.                 cv2.destroyAllWindows()

```